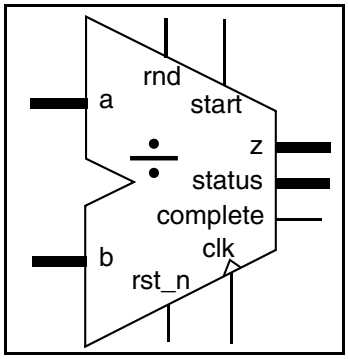
DW_fp_div_seq

Floating-Point Sequential Divider

Version, STAR, and myDesignWare Subscriptions: IP Directory

Features and Benefits

Revision History



- The precision format is parameterizable for either IEEE single, double precision, or a user-defined custom format
- Hardware for denormal numbers of IEEE 754 standard is selectively provided
- Accuracy conforms to the IEEE 754 standard
- Parameterized number of clock cycles
- Registered or un-registered input and outputs
- Internal register for the partial pipelining
- DesignWare datapath generator is employed for better timing and area.
- Includes a low-power implementation (at a sub-level) that has power benefits from minPower optimization (for details, see Table 1-3 on page 3)

Description

DW_fp_div_seq is a floating point sequential divider designed for small area, area-time trade-off, or high frequency (small clock period) applications. Input signals related to the timing control such as `clk`, `rst_n` and `start` are introduced, in addition to the output signal, `complete`. The final result of DW_fp_div_seq is equivalent to the result of DW_fp_div.

Component pins are described in Table 1-1 and configuration parameters are described in Table 1-2.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
a	(*exp_width* + *sig_width* + 1) bits	Input	Dividend
b	(*exp_width* + *sig_width* + 1) bits	Input	Divisor
rnd	3 bits	Input	Rounding mode; supports all rounding modes described in the *Datapath Floating-Point Overview*
clk	1 bit	Input	Clock
rst_n	1 bit	Input	Reset (active low)
start	1 bit	Input	Start operation
z	(*exp_width* + *sig_width* + 1) bits	Output	Division result

Table 1-1 Pin Description (Continued)

Pin Name	Width	Direction	Function
status	8 bits	Output	■ Status flags corresponding to `z`; for details, see STATUS Flags in the *Datapath Floating-Point Overview* ■ status[7]: Indicates divide-by-zero operation
complete	1 bit	Output	Completion flag

Table 1-2 Parameter Description

Parameter	Values	Description
sig_width	2 to 253 bits Default: 23	Word length of fraction field of floating-point numbers `a`, `b`, and `z`
exp_width	3 to 31 bits Default: 8	Word length of biased exponent of floating-point numbers `a`, `b`, and `z`
ieee_compliance	0 or 1 Default: 0	Level of support for IEEE 754: ■ 0: No support for IEEE 754 NaNs and denormals; NaNs are considered infinities and denormals are considered zeros ■ 1: Fully compliant with the IEEE 754 standard, including support for NaNs and denormals For more, see IEEE 754 Compatibility in the *Datapath Floating-Point Overview*.
num_cyc	4 to 2**sig_width* + 3 Default: 4	User-defined number of clock cycles to produce a valid result. The number depends on various parameters described in "Formula" on page 4.
rst_mode	0 or 1 Default: 0	Synchronous / asynchronous reset ■ 0: Asynchronous reset ■ 1: Synchronous reset
input_mode[a]	0 or 1 Default: 1	Input register setup ■ 0: No input register ■ 1: Input registers are inserted
output_mode	0 or 1 Default: 1	Output register setup ■ 0: No output register ■ 1: Output register is inserted
early_start	0 or 1 Default: 0	Computation start (only when *input_mode* is 1) ■ 0: Start computation in the second clock cycle ■ 1: Start computation in the first clock cycle Note: *early_start* should be 0 when *input_mode* is 0.

Table 1-2 Parameter Description (Continued)

Parameter	Values	Description
internal_reg	0 or 1 Default: 1	Internal register enables the pipeline operation (see Figure 1-1 on page 5) ■ 0: No internal register ■ 1: Internal register is inserted

a. When configured with the parameter *input_mode* set to '0', the inputs `a` and `b` MUST be held constant from the time `start` is asserted until `complete` has gone high to signal completion of the calculation. Conversely, if the parameter *input_mode* is set to '1', the `a` and `b` inputs are captured when `start` is high and, otherwise, ignored.

Table 1-3 Synthesis Implementations

Implementation Name	Function	License Feature Required
rtl[a]	Synthesis model	DesignWare (P-2019.03 and later) DesignWare-LP (before P-2019.03)

a. To achieve low-power benefits in sub-module implementations, you need to enable minPower; for details, see "Enabling minPower" on page 11.

Table 1-4 Simulation Models[a]

Model	Function
DW02.DW_FP_DIV_SEQ_SIM	Design unit name for VHDL simulation
dw/dw02/src/DW_fp_div_seq_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_fp_div_seq.v	Verilog simulation model source code

a. Note that during the computation phase (after `start` and before `complete` is asserted), the simulation models output X values and therefore cannot be used to compare with gate-level simulation.

DW_fp_div_seq has nine parameters which consist of three parameters (*sig_width*, *exp_width* and *ieee_compliance*) for the floating-point operation, one parameter (*rst_mode*) for the reset control, and four parameters (*num_cyc*, *input_mode*, *output_mode* and *early_start*) for the timing control. The actual number of clock cycles depends on the number of cycles defined by *num_cyc* and the use of input, output or internal registers. The following section, as well as Table 1-5 on page 7, describes the relationship between parameters and the actual number of clock cycles; that is, latency. The *early_start* parameter enables bypassing the input register when `start` is set to 1. It reduces the total latency by one clock cycle.

The *input_mode* parameter determines whether the inputs are to be registered inside DW_fp_div_seq (*input_mode* = *1*) or not (*input_mode* = *0*). If configured without input registers (*input_mode* = *0*), the logic that drives the inputs `a` and `b` must hold the input values constant for the entire time it takes to calculate the result (from the cycle before `start` drops until `complete` goes high). When configured with input registers (*input_mode* = *1*), the inputs `a` and `b` are captured when `start` is high and ignored until `start` goes high again.

When configured with no input registers, changes on inputs `a` and `b` while `complete` is low (calculation cycle) will produce unpredictable output values. Simulation models will produce unknown output values (Xs) and post an error message indicating the instance that violated this rule and the simulation time when the violation was detected.

The *output_mode* parameter determines whether the outputs are registered (*output_mode* = *1*) or not (*output_mode* = *0*).

Formula

The actual number of clock cycles required by a computation is calculated using the following formula:

```
Latency = actual number of cycles
        = num_cyc - (1 - output_mode) - (1 - input_mode) - early_start + internal_reg
```

The minimum number of cycles required for continuous operations is `num_cyc` - 1.

Block Diagram

Figure 1-1 describes the block diagram of DW_fp_div_seq. There are parameters that control the registers: *input_mode* and *output_mode* place the input and output registers; *early_mode* makes a bypass from the input of the input register to the output of the input register; *internal_reg* enables the partial pipeline operation.

Because the integer sequential divider consumes multiple clock cycles, the input data cannot be pipelined during the operation of the integer sequential divider. Once the integer sequential divider finishes the operation and passes the result to the normalization block, it can receive the next data from the input. Therefore, the final clock cycle of the current operation can be overlapped with the next operation as shown in Figure 1-2 on page 5.

Figure 1-1 Block Diagram of DW_fp_div_seq

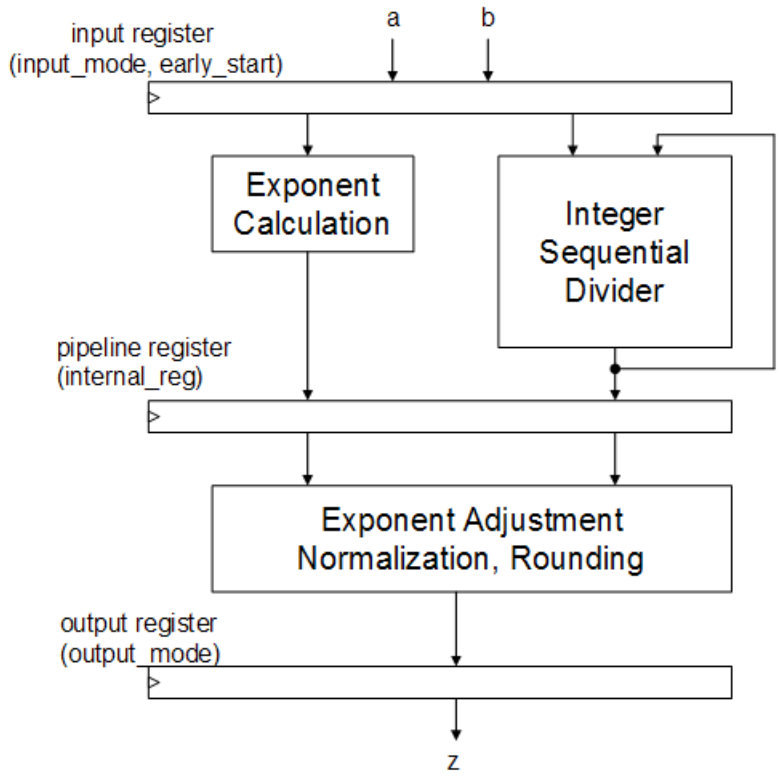
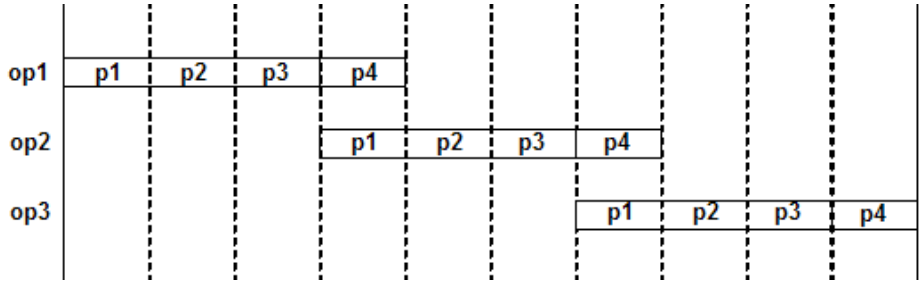


Figure 1-2 Pipeline Operation when *internal_reg* = 1 (*input_mode* = 0 and *output_mode* = 0)



Suppressing Warning Messages During Verilog Simulation

The Verilog simulation model includes macros that allow you to suppress warning messages during simulation.

To suppress all warning messages for all DWBB components, define the DW_SUPPRESS_WARN macro in either of the following ways:

- Specify the Verilog preprocessing macro in Verilog code:

  ```
  `define DW_SUPPRESS_WARN
  ```

- Or, include a command line option to the simulator, such as:

  `+define+DW_SUPPRESS_WARN` (which is used for the Synopsys VCS simulator)

The warning messages for this model include the following:

- If values other than 1 or 0 are present on a clock port, the following message is displayed:

  ```
  WARNING: <instance_path>.<clock_name>_monitor:
      at time = <timestamp>, Detected unknown value, x, on <clock_name> input.
  ```

  To suppress only this warning message for all DWBB components, use the following macro:

  - Define the DW_DISABLE_CLK_MONITOR macro. You can define this macro in the following ways:
    - Specify the Verilog preprocessing macro in Verilog code:

      ```
      `define DW_DISABLE_CLK_MONITOR
      ```

    - Or, include a command line option to the simulator, such as:

      `+define+DW_DISABLE_CLK_MONITOR` (which is used for the Synopsys VCS simulator)

  This message is also suppressed using the DW_SUPPRESS_WARN macro explained earlier.

- If an invalid rounding mode has been detected on `rnd`, the following message is displayed:

  ```
  WARNING: <instance_path>:
      at time = <timestamp>: Illegal rounding mode.
  ```

  To suppress this message, use the DW_SUPPRESS_WARN macro explained earlier.

- If the component is configured without an input register and an input operand changes during calculation, the following message is displayed:

  ```
  WARNING: <instance_path>:
      at time = <timestamp>, Operand input change on DW_fp_div_seq during calculation
  (configured without an input register) will cause corrupted results if operation is
  allowed to complete.
  ```

  To suppress this message, use the DW_SUPPRESS_WARN macro explained earlier.

Timing Waveforms

Based on the timing related parameters, *input_mode, output_mode*, *early_start* and *internal_reg*, there are 12 possible combinations of parameters, as shown in Table 1-5. When *input_mode* is 0, the *early_start* parameter has no meaning, so it should be kept at zero.

Table 1-5 Parameter Combinations

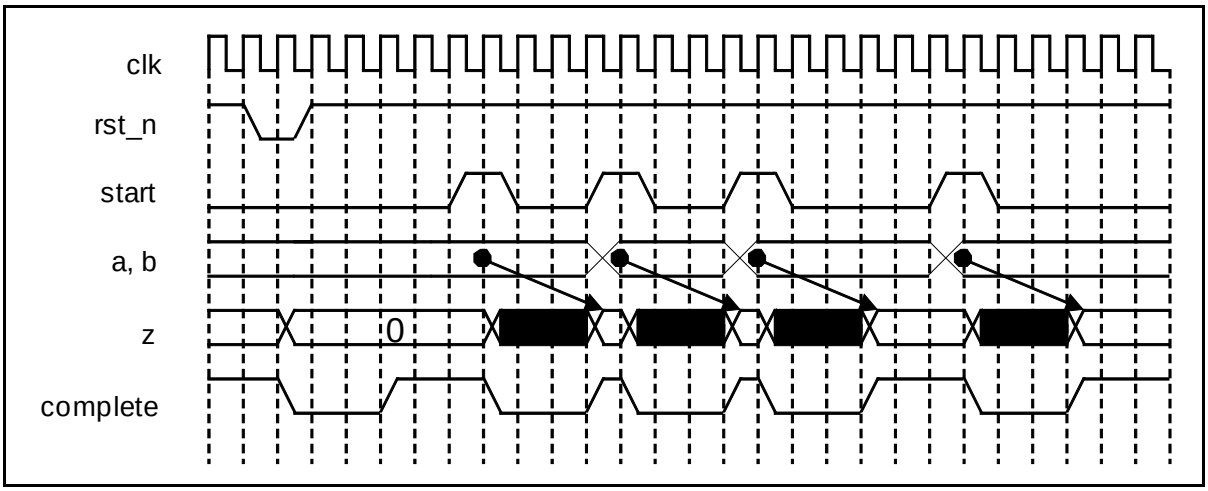
input_mode	output_mode	early_start	internal_reg	latency
0	0	0	0	*num_cyc* - 2
0	0	0	1	*num_cyc* - 1
0	0	1	0	N/A
0	0	1	1	N/A
0	1	0	0	*num_cyc* - 1
0	1	0	1	*num_cyc*
0	1	1	0	N/A
0	1	1	1	N/A
1	0	0	0	*num_cyc* - 1
1	0	0	1	*num_cyc*
1	0	1	0	*num_cyc* - 2
1	0	1	1	*num_cyc* - 1
1	1	0	0	*num_cyc*
1	1	0	1	*num_cyc* + 1
1	1	1	0	*num_cyc* - 1
1	1	1	1	*num_cyc*

Six timing diagrams are introduced assuming *internal_reg* is 1. When *internal_reg* is off, the valid output data are generated one clock cycle earlier than the timing diagram with *internal_reg* = 1

1. ***input_mode = 0, output_mode = 0, early_start = 0, internal_reg = 1 and num_cyc = 5***

   In this case, the input and output registers are not implemented. Any input change can affect the output result, so the input data should be kept unchanged by the end of the operation, and the next input data need to be changed with the toggled-on `start` signal. The `start` signal should be 'on' just for one clock cycle. After start signal initiates the division operation, the valid output `z` will appear after (*num_cyc* - 1) clock cycles.

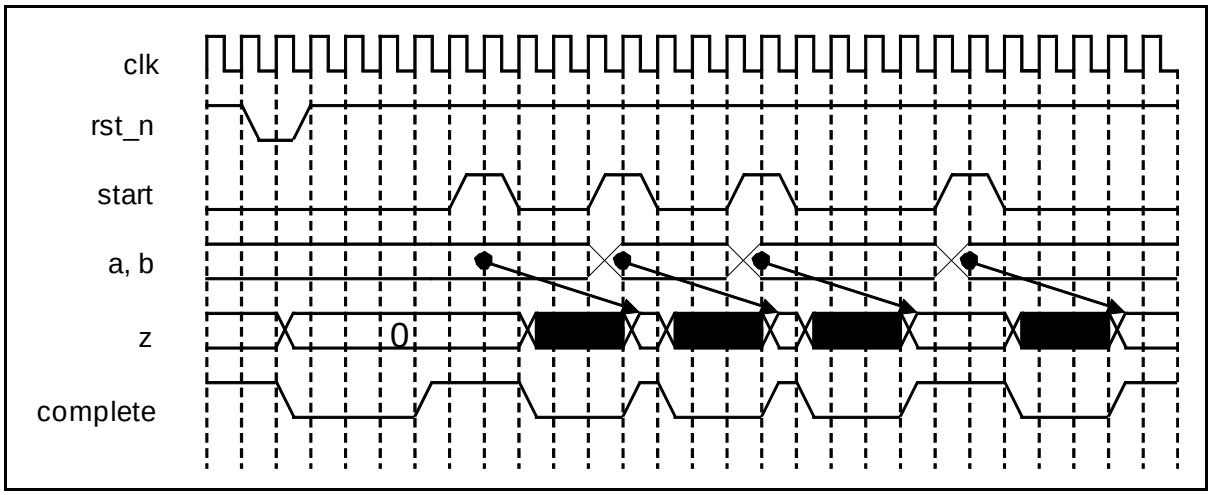
Figure 1-3 Simulation Waveform 1



2. ***input_mode = 0, output_mode = 1, early_start = 0, internal_reg = 1 and num_cyc = 5***

   An output register is implemented and the output result is valid after *num_cyc* clock cycle. The required timing operations of input data and `start` signal are same as the previous example: the input data should to be kept unchanged by the end of the operation, the next input data need to be changed with the toggled-on `start` signal and `start` signal should be 'on' just for one clock cycle.

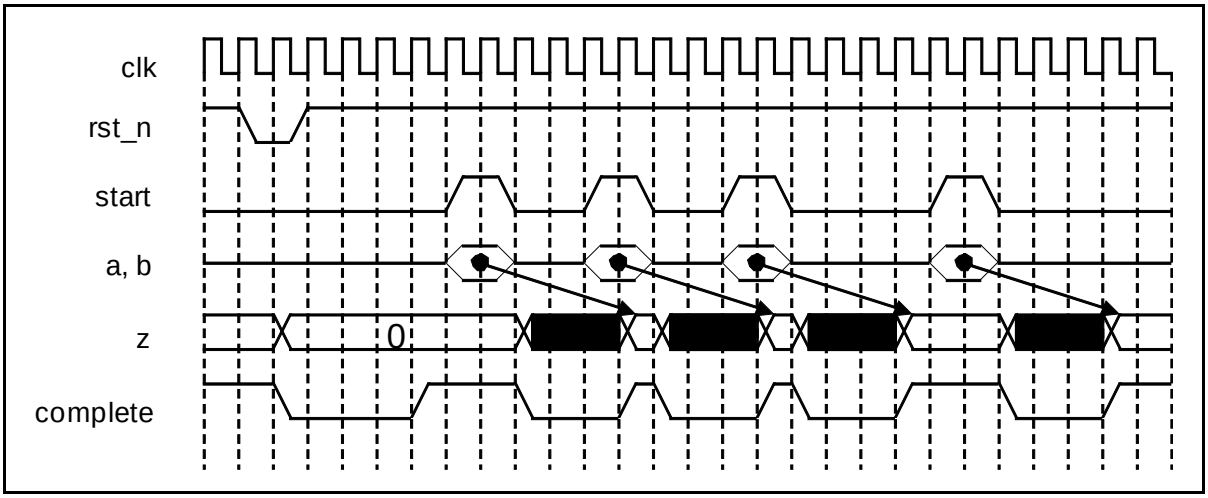
Figure 1-4 Simulation Waveform 2



3. ***input_mode = 1, output_mode = 0, early_start = 0, internal_reg = 1 and num_cyc = 5***

   If *input_mode* is on, the input registers hold the input data when `start` signal is activated. Any input data when `start` is off do not affect the division operation. The valid output is available after *num_cyc* clock cycles.

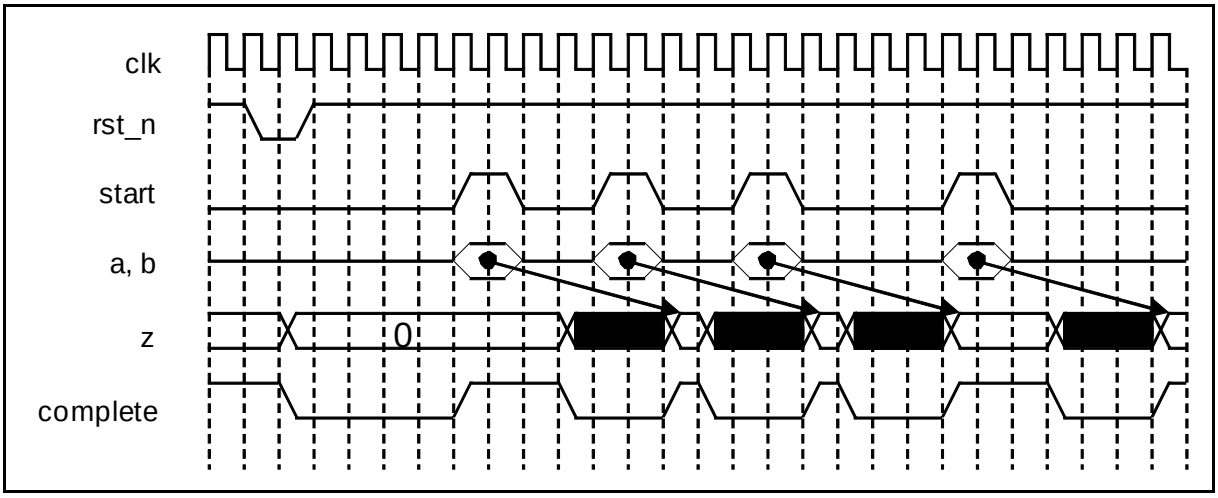
Figure 1-5 Simulation Waveform 3



4. ***input_mode = 1, output_mode = 1, early_start = 0, internal_reg = 1 and num_cyc = 5***

   Both input and output registers are implemented at the input and output ports, respectively. The timing diagram in Figure 1-6 shows that it has the same timing operation as Figure 1-5 except that the output data has one additional delay due to the output register. Therefore, the latency becomes (*num_cyc* + 1) clock cycles.

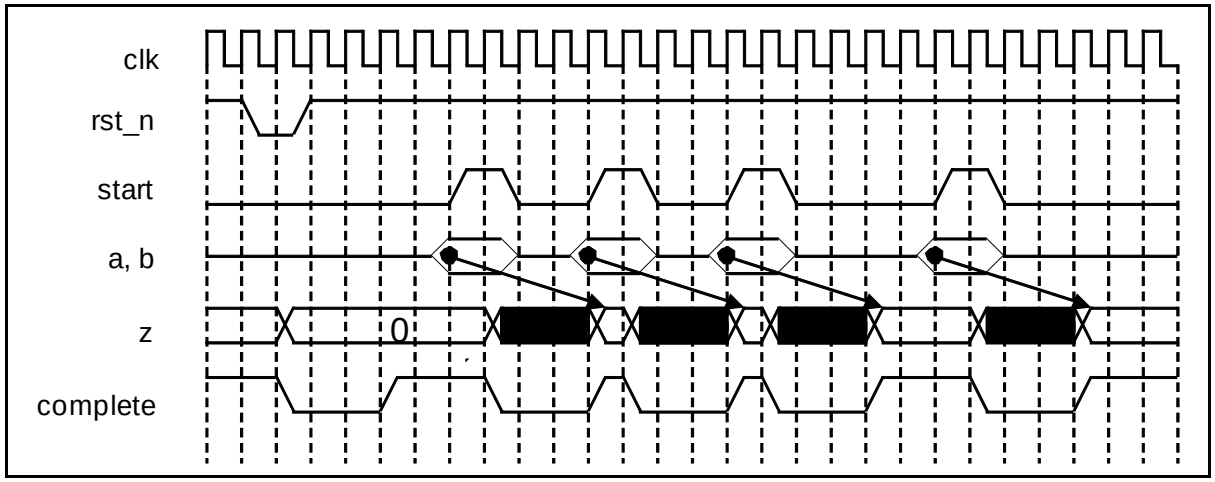
Figure 1-6 Simulation Waveform 4



5. ***input_mode = 1, output_mode = 0, early_start = 1, internal_reg = 1 and num_cyc = 5***

   For this case, *early_start* is turned on and *early_start* contributes to reduce one clock cycle of the total latency by receiving the input data one clock earlier than normal operations. In this case, the valid output appears after (*num_cyc* - 1) clock cycles.

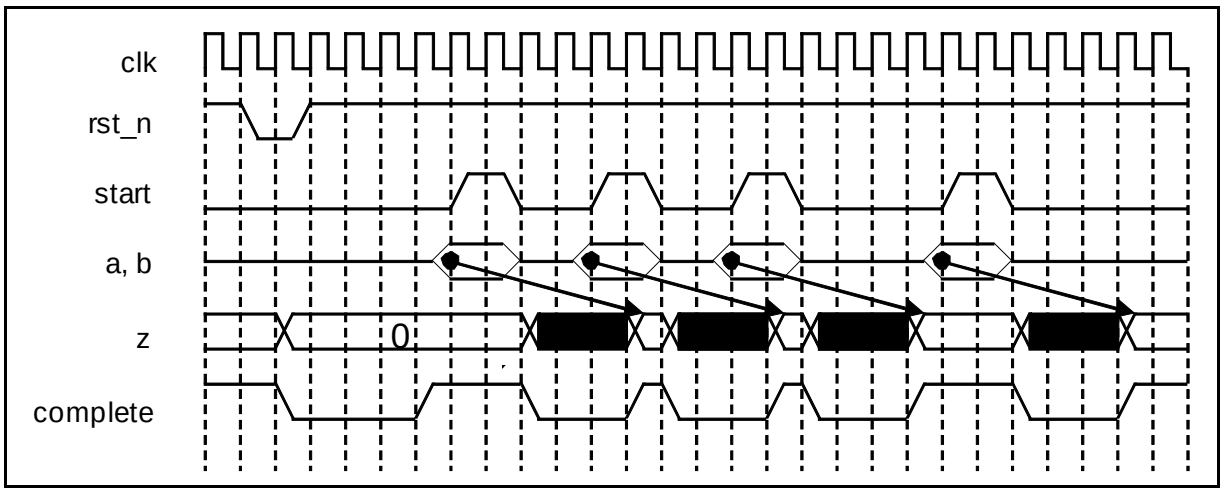
Figure 1-7 Simulation Waveform 5



6. ***input_mode = 1, output_mode = 1, early_start = 1, internal_reg = 1 and num_cyc = 5***

   All parameters are on, so input and output registers are implemented in addition to the input bypass. The timing diagram in Figure 1-8 has a similar shape to Figure 1-7, but the valid outputs appear with one additional clock cycle delay, so the latency of this case becomes *num_cyc* clock cycles.

Figure 1-8 Simulation Waveform 6



DW_fp_div_seq by default provides the hardware for denormal numbers and NaNs as defined in the IEEE 754 standard. If the parameter *ieee_compliance* is turned off, denormal numbers are considered as zeros, and NaNs are considered as Infinity. Otherwise, denormal numbers and NaNs become effective and additional hardware for denormal numbers is integrated.

For more information about floating-point, including `status` flag bits, and integer and floating-point formats, refer to the *Datapath Floating-Point Overview*.
sc_uint < *sig_width* + *exp_width* + 1 >

```
DW_fp_div_seq< sig_width, exp_width, ieee_compliance, num_cyc, rst_mode, \
input_mode, output_mode, early_start, internal_reg >::implement  (
    sc_uint< sig_width+exp_width+1 >  a,
    sc_uint< sig_width+exp_width+1 >  b,
    sc_uint< 3 >  rnd,
    sc_uint< 1 >  clk,
    sc_logic  rst_n,
    sc_uint< 1 >  start,
    sc_uint< 8 > *  status,
    sc_uint< 1 > *  complete )
```

Enabling minPower

You can instantiate this component without enabling minPower, but to achieve power savings from the low-power implementation (at a sub-level--see Table 1-3 on page 3), you must enable minPower optimization, as follows:

- Design Compiler
  - Version P-2019.03 and later:

    ```
    set power_enable_minpower true
    ```

  - Before version P-2019.03 (requires the DesignWare-LP license feature):

    ```
    set synthetic_library {dw_foundation.sldb dw_minpower.sldb}
    set link_library {* $target_library $synthetic_library}
    ```

- Fusion Compiler

  Optimization for minPower is enabled as part of the total_power metric setting. To enable the total_power metric, use the following:

  ```
  set_qor_strategy -stage synthesis -metric total_power
  ```

Related Topics

- Datapath Floating-Point Overview

HDL Usage Through Component Instantiation - VHDL

```
library IEEE,DWARE;
use IEEE.std_logic_1164.all;
use DWARE.DWpackages.all;
use DWARE.DW_Foundation_comp_arith.all;

entity DW_fp_div_seq_inst is
    generic (
     inst_sig_width : POSITIVE := 23;
     inst_exp_width : POSITIVE := 8;
     inst_ieee_compliance : INTEGER := 0;
     inst_num_cyc : POSITIVE := 4;
     inst_rst_mode : INTEGER := 0;
     inst_input_mode : INTEGER := 1;
     inst_output_mode : INTEGER := 1;
     inst_early_start : INTEGER := 0;
     inst_internal_reg : INTEGER := 1
     );
    port (
     inst_a : in std_logic_vector(inst_sig_width+inst_exp_width downto 0);
     inst_b : in std_logic_vector(inst_sig_width+inst_exp_width downto 0);
     inst_rnd : in std_logic_vector(2 downto 0);
     inst_clk : in std_logic;
     inst_rst_n : in std_logic;
     inst_start : in std_logic;
     z_inst : out std_logic_vector(inst_sig_width+inst_exp_width downto 0);
     status_inst : out std_logic_vector(7 downto 0);
     complete_inst : out std_logic
     );
end DW_fp_div_seq_inst;


architecture inst of DW_fp_div_seq_inst is

begin

    -- Instance of DW_fp_div_seq
    U1 : DW_fp_div_seq
    generic map (
          sig_width => inst_sig_width,
          exp_width => inst_exp_width,
          ieee_compliance => inst_ieee_compliance,
          num_cyc => inst_num_cyc,
          rst_mode => inst_rst_mode,
          input_mode => inst_input_mode,
          output_mode => inst_output_mode,
          early_start => inst_early_start,
          internal_reg => inst_internal_reg
          )
```

```
      port map (
            a => inst_a,
            b => inst_b,
            rnd => inst_rnd,
            clk => inst_clk,
            rst_n => inst_rst_n,
            start => inst_start,
            z => z_inst,
            status => status_inst,
            complete => complete_inst
            );


  end inst;

  -- pragma translate_off
  configuration DW_fp_div_seq_inst_cfg_inst of DW_fp_div_seq_inst is
  for inst
  end for; -- inst
  end DW_fp_div_seq_inst_cfg_inst;
  -- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_fp_div_seq_inst( inst_a, inst_b, inst_rnd, inst_clk, inst_rst_n,
          inst_start, z_inst, status_inst, complete_inst );

parameter inst_sig_width = 23;
parameter inst_exp_width = 8;
parameter inst_ieee_compliance = 0;
parameter inst_num_cyc = 4;
parameter inst_rst_mode = 0;
parameter inst_input_mode = 1;
parameter inst_output_mode = 1;
parameter inst_early_start = 0;
parameter inst_internal_reg = 1;


input [inst_sig_width+inst_exp_width : 0] inst_a;
input [inst_sig_width+inst_exp_width : 0] inst_b;
input [2 : 0] inst_rnd;
input inst_clk;
input inst_rst_n;
input inst_start;
output [inst_sig_width+inst_exp_width : 0] z_inst;
output [7 : 0] status_inst;
output complete_inst;

    // Instance of DW_fp_div_seq
    DW_fp_div_seq #(inst_sig_width, inst_exp_width, inst_ieee_compliance, inst_num_cyc,
inst_rst_mode, inst_input_mode, inst_output_mode, inst_early_start, inst_internal_reg)
U1 (
          .a(inst_a),
          .b(inst_b),
          .rnd(inst_rnd),
          .clk(inst_clk),
          .rst_n(inst_rst_n),
          .start(inst_start),
          .z(z_inst),
          .status(status_inst),
          .complete(complete_inst) );

endmodule
```

Revision History

For notes about this release, see the *DesignWare Building Block IP Release Notes*.

For lists of both known and fixed issues for this component, refer to the STAR report.

For a version of this datasheet with visible change bars, click here.

Date	Release	Updates
July 2020	DWBB_201912.5	■ Adjusted the description of the *ieee_compliance* parameter in Table 1-2 on page 2 ■ Changed title of section "Suppressing Warning Messages During Verilog Simulation" on page 6 and added ability to globally suppress all warning messages
April 2020	DWBB_201912.3	■ Updated the description of the `status` port in Table 1-1 on page 1 ■ Updated the value range of *sig_width* in Table 1-2 on page 2
October 2019	DWBB_201903.5	■ Added the "Disabling Clock Monitor Messages" section
March 2019	DWBB_201903.0	■ Clarified license requirements in Table 1-3 on page 3 ■ Added "Enabling minPower" on page 11 ■ Added this Revision History table and the document links on this page

Copyright Notice and Proprietary Information

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.

All other product or company names may be trademarks of their respective owners.

Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
www.synopsys.com